CMSC 22200 Computer Architecture
The University of Chicago Autumn 2015
Lab Assignment 2
Due: Tuesday, November 10 at the beginning of class by hardcopy

In this lab assignment, we explore the effects of static and dynamic pipelining techniques using the Marss simulator. By dynamic, we are referring to superscalar in-order or out-of-order pipelines.

You will continue to use QEMU and your installation and disk image from Lab 1. Refer back to the writeup for Lab 1 for details on installation and use.

Effects of Different Pipelining Techniques
You will run Parsec benchmarks on two different cores; one of them with a static pipeline and the other with a dynamic pipeline. The goal is to observe the differences in performance of those cores.
1. Run Blackscholes and Swaptions from Parsec for 200m instructions on the atom_core machine (if you still have your results from Lab 1, you may refer back to them instead) and answer the following questions:
Q1. How many clock cycles are taken for each of the benchmarks?
Q2. How many instructions are committed (note: committed, not issued) to finish each of the benchmarks?
Q3. Calculate IPC given the answers for Q1 and Q2 for each of the benchmarks. (Remember that we briefly mentioned in class that IPC is instructions per cycle, the reciprocal of CPI that we sometimes use when we have a superscalar machine and CPI < 1. But, note that this is not a superscalar machine.)
Q4. Is the IPC you calculated the same as the one listed in the commit section?
Q5. Look at the fetch stage: how many instructions are fetched for each of the benchmarks?
Q6. Why is the number of I-cache accesses different than the total number of instructions fetched? What is the average ratio?
2. Run the same benchmarks on the single_core machine, which is a superscalar and Out-Of-Order architecture. Answer the following questions:
Q7. How many clock cycles are taken for each of the benchmarks?
Q8. How many instructions are committed to finish each of the benchmarks? Is it almost the same as the number for atom_core?
3. Both machines ran the same programs, and use the same instructions set.
Q9. Compute the CPI for both machines. What does this number mean for how many instructions the machine can execute at a time?
Q10. Comment on the relative execution efficiency of these x86 implementations.
Q11. If they had the same clock rate, which one would be faster? By

how much?

4. Recall that a key technique to make the CISC x86 instruction set fast is dynamic translation of the x86 instructions to simple u-ops.
Q12. Find the number of u-ops executed in each of the two implementations in the statistics file. Is it different for the two machines? What can you conclude from this?
5. Another key technique to make a CISC x86 instruction run fast is fast decode of “simple” instructions, and slower decode for the larger, more complex instructions. Find in the trace file information about the “complex” and “fast” decode types.
Q13. What fraction of the instructions involves “fast” (simple) decodes? Is it similar for the two architectures?
Q14. Do any of the instructions require the use of microcode on single_core?
6. In the single_core machine, there are many instructions in execution at the same time. As the numbers of instructions executed simultaneously increases, an increasing number of values are read from the “forwarding circuits”, or as Intel refers to them, the “reorder buffer” or ROB and the rename table.
Q15. Find the number of values read and written from the reorder buffer and the rename table.
Q16. Compare these to the number of “physreg_reads” and “physreg_writes”. What fraction of the values used by the computation actually come from the physical register file, as opposed to from forwarding?
7. Both the atom_core and single_core implementations do branch prediction.
Q17. What are the success rates for branch prediction? Is this a high or low number?
Q18. Does this make the program look more like a bunch of short code segments separated by branches, or like a single long stream of instructions to the processor?
Q19. Compute the average number of instructions between mispredicted branches.